The Parallel C?? "vergence"

and the Split-C Perspective

David E. Culler

Computer Science Division

University of California, Berkeley

Supercomputing 93

Panel on Parallel C

Required Inflammatory Remarks

- **Most interesting new parallel languages are *toys*.**
- **Toys are wonderful to play with, we need *lots* of interesting toys.**
  - but don't advertise graphs, trees, etc. and deliver square matric multiply
- **Don't converge into an ugly, amorphous, colorless, tasteless mass just because it seems politically or economically correct.**
  - the union of many mediocre languages is always worse than its parts. (The interesection is at least smaller.)
- ***Accentuate your differences.* That is what we learn from.**
- **If we are to have only one, let's be patient and allow a really good one to emerge.**

Pitfalls of Interesting Parallel Languages

(or why I leave toys in the yard to rust)

- **Space**
- **Time**
- **Expressibility**

Space

Don't double (quaduple?) the size of my already huge data set by adding hidden fields (`my position?`) or by making hidden copies.

Time

Don't double (quaduple?) the time to execute my program by preventing me from using highly optimized local routines.

Don't tell me the only way to optimize my program is to wait for a better compiler to come out.

Don't force me to second-guess the compiler transformations in order to do what I wan't done.

Don't get in my way!

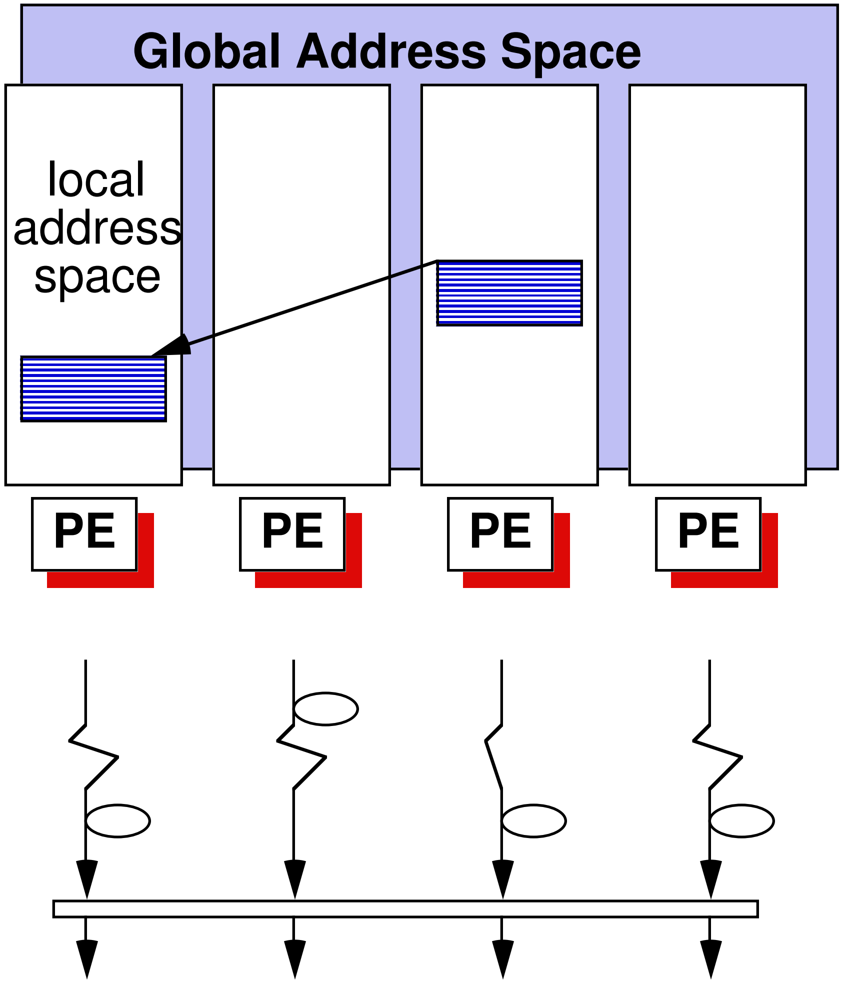
Expressiveness

Don't double the length of my program with obscure declarations and tell me how much "simpler and clearer" it is.

Don't make me convolute my algorithms and data structures into the particular abstractions that you, the language developer, like. I will build my own abstractions, thank you.

Don't take away my ponters!

Split-C



- **gcc C on every node, *plus***
- **Global pointers**

  2D global address space
- **Powerful global assignment**

  `:=` (get, put)

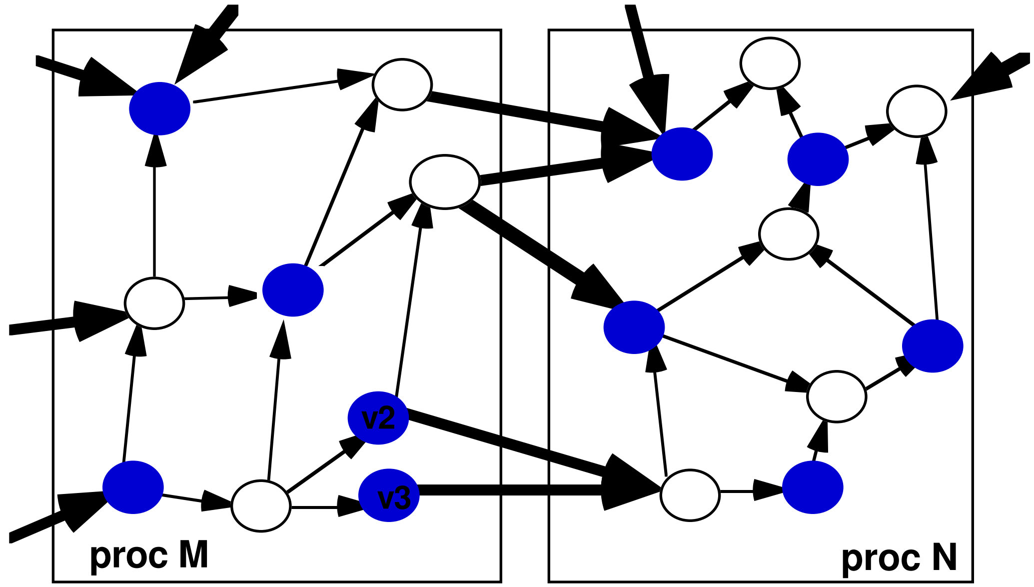
  `:-` (store)

  bulk read,write,get,put,store
- **Canonical storage layout**

  `A[m][n]::[b][b];`

Example: Owner Computes on a Bipartite Graph

```
typedef struct node_t {
    double value;
    int edge_count;
    double *coeffs;
    double *global (*values);
    struct node_t *next;
} node_t;
```



data parallel
by convention

```
void all_compute_e()
{
    node_t *n;
    int i;
    for (n = e_nodes; n; n = n->next) {
      for (i = 0; i < n->edge_count; i++)
         n->value = n->value - *(n->values[i]) * (n->coeffs[i]);
    }
   barrier();
}
```

List of local nodes

Global read

Engineering for Performance within the Language

Split Phase Access

Get: Initiate reads to locals, continue computing, sync before use

```
lx := *Gx;
ly := *Gy;
. . .;
sync();
z = lx + ly;
```

Put: Initiate write from locals and continue computing.

– consistency ensured only at sync

```
*Gx := lx+ly;
*Gy := Gy;
. . .;
sync();
```

Store: Initiate write signal destination of completion

```
*Gx :- lx+ly;
all_store_sync();          (or store_sync(n);)
```

Canonical Storage Layout

Matrix

1,1	1,2	1,3	1,4
2,1	2,2	2,3	2,4
3,1	3,2	3,3	4,4

Easy to specify traversal given canonical layout

C Row major

1,1
1,2
1,3
1,4
2,1
2,2
2,3
2,4
3,1
3,2
3,3
4,4

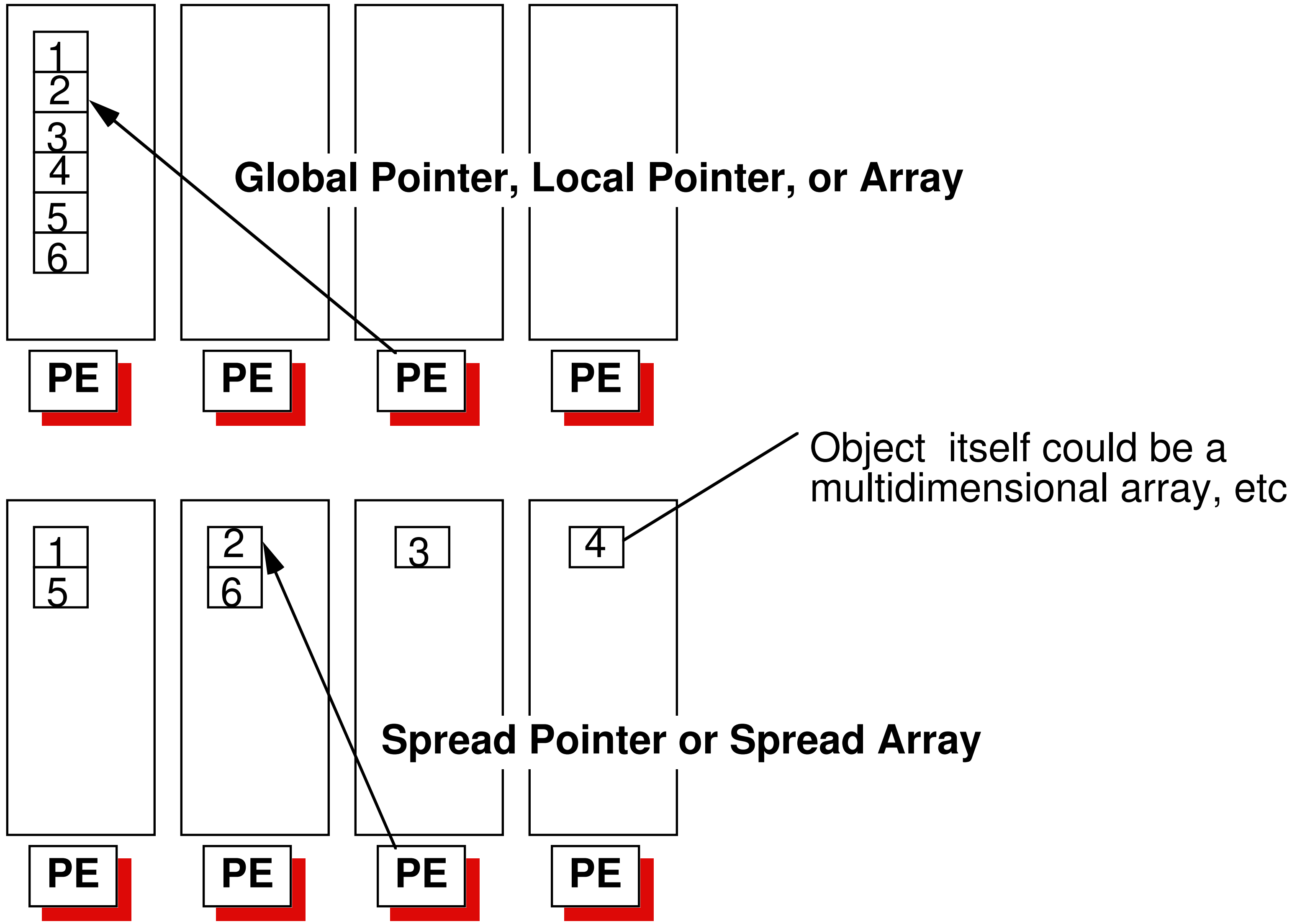
+1

+C

Fortran Column Major

1,1
2,1
3,1
1,2
2,2
3,2
1,3
2,3
3,3
1,4
2,4
4,4

Split-C Storage Layout



Spread Arrays

Spread Arrays are spread over the entire machine in a uniform rectangle

```
foo A[n][r]::[b][b],
```

linearize high dimensions
under cyclic map

sizeof object = b * b * sizeof(foo)

Simple to generate code.

Simple to iterate over the local elements of a spread array.

The traditional C duality between arrays and pointers is preserved.
spread arrays => spread pointers :: global pointers

Blocked Matrix Multiply

```
void all_mat_mult_blk(int n, int r, int m, int b,
                      double C[n][m]::[b][b],
                      double A[n][r]::[b][b],
                      double B[r][m]::[b][b])
{
  int i,j,k,l;
  double la[b][b], lb[b][b];        /* Local copies of the subblocks */
  for_my_2D(i,j,l,n,m) {
    double (*lc)[b] = tolocal(C[i][j]);
    for (k=0;k<r;k++) {
      bulk_read (la, A[i][k], b*b*sizeof(double));
      bulk_read (lb, B[k][j], b*b*sizeof(double));
      matrix_mult(b,b,b,lc,la,lb);
    }
  }
  barrier();
}
```

Configuration independent use of spread arrays

Highly optimized local routine

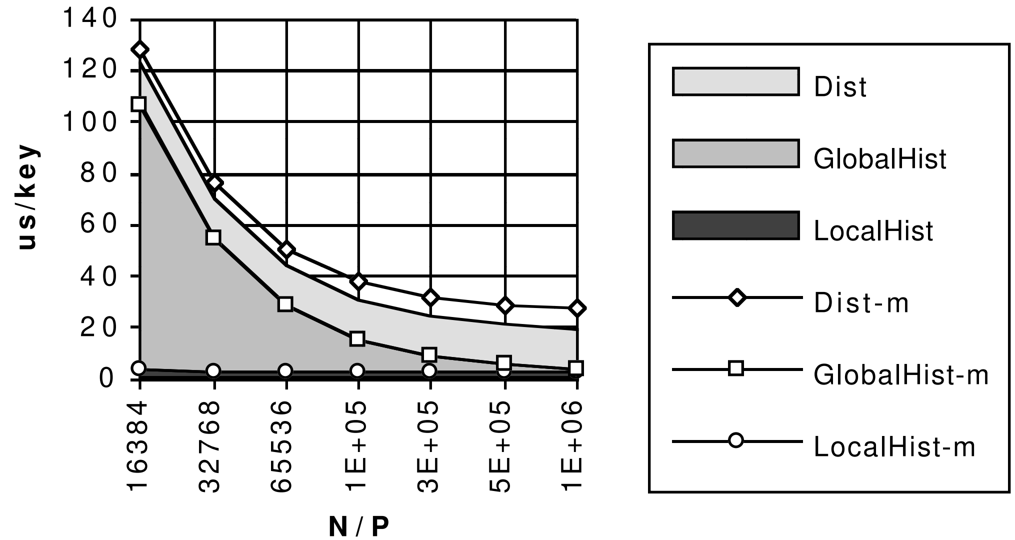
Data parallel by convention, not by decree.

Performance of Matrix Multiply

Tackling Real-Sized Problems

(Data size > 1/2 of memory)

Parallel Hist-Radix Sort
512 proc.
up to 1/2 billion keys



Sample Sort
512 proc.
up to 1/2 billion keys